FaceMap PRO - Documentação Técnica

RESUMO

Este trabalho apresenta o desenvolvimento do FaceMap PRO, um sistema *desktop* de reconhecimento facial que demonstra na prática o funcionamento de vetores de características (*face descriptors*) utilizados em biometria facial. Implementado em Python com *OpenCV* e modelos de *deep learning* (*YuNet* e *SFace*), o sistema ilustra de forma visual e interativa como redes neurais convolucionais extraem e comparam características faciais únicas através de representações numéricas de 128 dimensões. O objetivo principal é tornar tangível o processo matemático de reconhecimento facial, demonstrando conceitos como extração de *embeddings*, similaridade por cosseno, normalização de vetores e *tracking* temporal com votação.

Palavras-chave: Vetores de Características, *Face Descriptors*, Embeddings Faciais, Similaridade de Cosseno, *OpenCV, Deep Learning.*

1. INTRODUÇÃO

1.1 Contextualização

O reconhecimento facial moderno baseia-se na conversão de imagens faciais em representações matemáticas compactas chamadas de "*embeddings*" ou "*face descriptors*". Esses vetores, tipicamente com 128 ou 512 dimensões, encapsulam características faciais distintivas aprendidas por redes neurais profundas. O processo de reconhecimento torna-se então uma comparação de vetores numéricos ao invés de comparação direta de pixels.

1.2 Justificativa

Embora a literatura técnica sobre reconhecimento facial seja extensa, existe uma lacuna entre a teoria matemática e sua aplicação prática. Conceitos como "*embeddings* de 128 dimensões", "espaço de características" e "similaridade de cosseno" são frequentemente abstratos e de difícil visualização. Este projeto preenche essa lacuna criando uma ferramenta que mostra os números reais que compõem esses vetores, tornando visível um processo que normalmente ocorre de forma opaca dentro de redes neurais.

Objetivo Geral

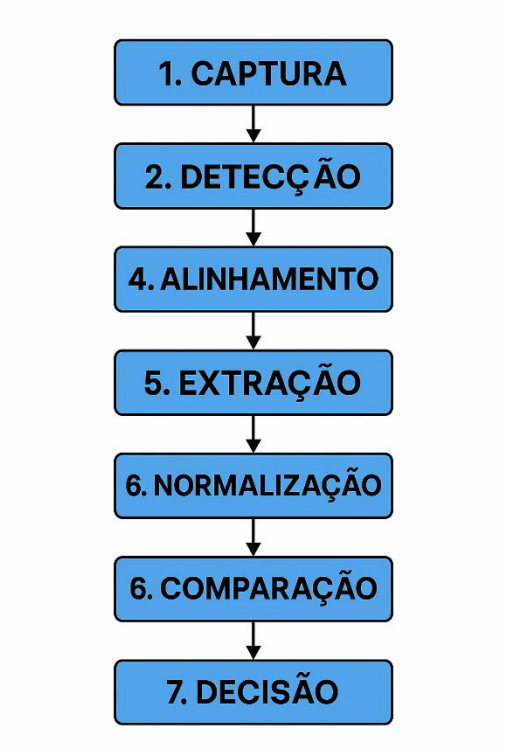
Desenvolver um sistema desktop interativo que demonstre na prática a extração, visualização e comparação de vetores de características faciais, tornando tangível o funcionamento matemático de sistemas de reconhecimento biométrico baseados em *deep learning*.

Objetivos Específicos

- Implementar extração de face *descriptors* de 128 dimensões usando redes neurais convolucionais
- Demonstrar visualmente os valores numéricos dos vetores e sua normalização L2
- Ilustrar o cálculo de similaridade por cosseno entre vetores faciais
- Implementar sistema de *tracking* com votação temporal para estabilização do reconhecimento
- Criar visualização em tempo real de *landmarks* e medidas biométricas

2. FUNCIONAMENTO DO ALGORITMO

2.1 Pipeline Completo do Sistema



2.2 Descrição de Cada Etapa

Etapa 1: Captura de Vídeo

O sistema captura frames da webcam em resolução 640x480 pixels a 30 FPS usando a biblioteca OpenCV. Os frames são processados continuamente em tempo real.

Etapa 2: Detecção de Faces

A rede neural ***YuNet*** analisa cada frame e retorna:

- ***Bounding box*** (x, y, largura, altura) de cada face detectada;
- **5 *landmarks* faciais**: olho direito, olho esquerdo, nariz, canto direito da boca, canto esquerdo da boca;
- ***Score* de confiança** da detecção;

Nota: Para otimizar performance, a detecção ocorre apenas a cada 8 frames. Nos frames intermediários, usa-se tracking.

Etapa 3: Alinhamento Facial

Usando os *landmarks* detectados, a face é rotacionada e recortada para uma pose padronizada. Isso garante que faces em diferentes ângulos sejam normalizadas antes da extração de características.

Etapa 4: Extração do *Embedding*

A rede neural ***SFace*** processa a face alinhada e gera um vetor de **128 números reais** (*float32*). Este vetor é o "*embedding*" ou "*face descriptor*" - uma representação compacta das características faciais únicas daquela pessoa.

Exemplo de *embedding (bruto)*:

"[-0.04968526214361191, -0.04072842001914978, 0.03125116601586342, 0.00028480537002906203, 0.2768245339393616, 0.059226926416158676...]"

└──────────── 128 valores ────────────┘

Etapa 5: Normalização L2 (norma Euclidiana)

O vetor é normalizado para ter comprimento (norma) exatamente igual a 1.0. Isso garante que a comparação seja baseada na direção do vetor, não em sua magnitude, tornando o sistema robusto a variações de iluminação.

Etapa 6: Comparação com Banco de Dados

O *embedding* normalizado é comparado com todos os *embeddings* cadastrados usando similaridade de cosseno. A métrica retorna valores entre -1.0 (vetores opostos) e +1.0 (vetores idênticos).

Etapa 7: Decisão de Reconhecimento

Para um reconhecimento ser considerado válido, duas condições devem ser satisfeitas:

- **Similaridade ≥ 0.65** (face é similar à cadastrada);
- ***Gap* ≥ 0.15** (diferença entre melhor e segundo melhor *match*);

O *gap* evita falsos positivos em situações ambíguas onde múltiplas pessoas têm similaridades próximas.

2.3 Sistema de Tracking Temporal

Entre detecções completas (que ocorrem a cada 8 frames), o sistema usa *object tracking* (algoritmos KCF/MOSSE/CSRT) para seguir as faces detectadas sem re-executar a detecção pesada.

Adicionalmente, implementa-se um sistema de votação que mantém um *buffer* dos últimos 25 frames. O nome exibido é decidido por maioria (mínimo 7 votos), eliminando oscilações e falsos positivos esporádicos.

Exemplo de votação:

Buffer: [João, João, ?, João, Maria, João, João, João, ...]

└──────────── últimos 25 frames ────────────┘

Contagem: João = 18 votos, Maria = 1 voto, Desconhecido = 1 voto

Decisão: "João" (≥ 7 votos, maioria clara)

3. FÓRMULAS MATEMÁTICAS

3.1 Normalização L2

A normalização garante que todos os vetores tenham o mesmo comprimento (norma = 1.0), eliminando variações de magnitude causadas por iluminação ou contraste.

Fórmulas:

Principal da normalização L2: $\hat{v} = v / ||v||_2$

Auxiliar (para calcular a norma L2): $||v||_2 = \sqrt{(v_1^2 + v_2^2 + v_3^2 + ... + v_n^2)}$

Onde:

- v = vetor original de 128 dimensões
- $\hat{v}$ = vetor normalizado
- $||v||_2$ = norma L2 (comprimento euclidiano do vetor)

Exemplo:

Fragmento de Vetor Original:

V =
[−0.0496852621, −0.0407284200, 0.0312511660, 0.0002848054, 0.2768245339, 0.0592269264]

Cálculo:

$||v||_2 = \sqrt{((-0.0496852621)^2 + (-0.0407284200)^2 + (0.0312511660)^2 + (0.0002848053)^2 + (0.2768245339)^2 + (0.0592269264)^2)}$

$||v||_2 = \sqrt{(0.002468 + 0.001658 + 0.000976 + 0.000000081 + 0.076621 + 0.003508)}$

$||v||_2 = \sqrt{(0.085231)}$

$\mathbf{||v||_2 \approx 0.2919}$

$\hat{v} = [-0.0496852621 / 0.2919] = \mathbf{-0.170175}$

$\hat{v} = [-0.0407284200 / 0.2919] = \mathbf{-0.139497}$

$\hat{v} = [0.0312511660 / 0.2919] = \mathbf{0.107037}$

$\hat{v} = [0.0002848053 / 0.2919] = \mathbf{0.000975}$

$\hat{v} = [0.2768245339 / 0.2919] = \mathbf{0.948142}$

$\hat{v} = [0.0592269264 / 0.2919] = \mathbf{0.202856}$

$||\hat{v}||_2 = \sqrt{((-0.1701)^2 + (-0.1394)^2 + 0.1070^2 + 0.0009^2 + 0.9481^2 + 0.2028^2)} \approx \mathbf{1.0}$

3.2 Similaridade de Cosseno

A similaridade de cosseno mede o ângulo entre dois vetores normalizados. Quanto menor o ângulo, mais similares são as faces.

Fórmula:

$\cos(\theta) = \hat{v}_1 \odot \hat{v}_2 = \Sigma(\hat{v}_1[i] \times \hat{v}_2[i])$

(i=1 até 128)

Onde:

- $\hat{v}_1$ = *embedding* normalizado da face 1
- $\hat{v}_2$ = *embedding* normalizado da face 2
- $\odot$ = produto escalar (*dot product*)
- θ = ângulo entre os vetores
- $\cos(\theta)$ = similaridade (varia de -1 a +1)

Interpretação dos valores:

- $\cos(\theta) = 1.0 \rightarrow$ Ângulo 0° (vetores idênticos, mesma pessoa)
- $\cos(\theta) = 0.7 \rightarrow$ Ângulo ≈ 45° (faces similares, provável match)
- $\cos(\theta) = 0.0 \rightarrow$ Ângulo 90° (faces ortogonais, diferentes)
- $\cos(\theta) = -1.0 \rightarrow$ Ângulo 180° (vetores opostos)

Exemplo:

$\hat{v}_1 = [a_1, a_2, a_3, a_4, a_5, a_6]$

$\hat{v}_2 = [b_1, b_2, b_3, b_4, b_5, b_6]$

$\hat{v}_1 = [-0.170175, -0.139497, 0.107037, 0.000975, 0.948142, 0.202856]$

$\hat{v}_2 = [-0.160000, -0.140000, 0.100000, 0.001000, 0.950000, 0.200000]$

$\cos(\theta) = (-0.170175 \times -0.160000) + (-0.139497 \times -0.140000) + (0.107037 \times 0.100000) + (0.000975 \times 0.001000) + (0.948142 \times 0.950000) + (0.202856 \times 0.200000)$

$\cos(\theta) = (0.027228) + (0.019530) + (0.010704) + (0.0000009) + (0.900735) + (0.040571) \approx$ **0.9988 → rostos extremamente parecidos (mesma pessoa)**

3.3 Gap de Similaridade

O gap mede a diferença entre o melhor match e o segundo melhor, indicando confiança na decisão.

Fórmula:

gap = sim_melhor - sim_segundo_melhor

Onde:

- sim_melhor = maior similaridade encontrada no banco de dados
- sim_segundo_melhor = segunda maior similaridade

Critério de decisão:

if (sim_melhor ≥ 0.65) AND (gap ≥ 0.15):

reconhecimento_válido = True

else:

reconhecimento_válido = False

Exemplo:

Comparação com banco de dados:

- Pessoa A: sim = 0.78

- Pessoa B: sim = 0.42

- Pessoa C: sim = 0.31

gap = 0.78 - 0.42 = 0.36

Decisão:

- sim_melhor (0.78) ≥ 0.65

- *gap* (0.36) ≥ 0.15

→ *MATCH* COM PESSOA A

3.4 Votação Temporal

A votação estabiliza o reconhecimento ao longo do tempo, eliminando predições esporádicas incorretas.

Fórmula:

nome_final = argmax(count(buffer))

se count(nome_final) ≥ 7

Onde:

- *buffer* = deque circular com últimos 25 *frames*;
- *count*(x) = número de ocorrências de x no *buffer*;
- *argmax* = elemento com maior contagem;

Exemplo:

Buffer (25 posições): [A, A, ?, A, B, A, A, A, A, ?, A, A, A, A, A, ...]

Contagem:

- A: 18 votos;

- B: 1 voto;

- ?: 6 frames sem detecção;

Decisão: A (18 ≥ 7 votos mínimos)

4. CRITÉRIOS DE QUALIDADE NA CAPTURA

Para garantir que apenas imagens de boa qualidade sejam usadas no cadastro, o sistema avalia três métricas principais:

4.1 Tamanho Facial Mínimo

Critério: min (largura, altura) ≥ 100 *pixels*

Justificativa: Faces muito pequenas (distantes da câmera) resultam em baixa resolução de features como olhos e boca, prejudicando a extração de características.

Fórmula:

tamanho_ok = (w ≥ 100) AND (h ≥ 100)

Onde:

- w = largura da *bounding box* em *pixels*
- h = altura da *bounding box* em *pixels*

4.2 Distância Interocular (IO)

Critério: IO ≥ 28 pixels

Justificativa: A distância entre os olhos é um indicador robusto de que o rosto está de frente e próximo o suficiente da câmera. Faces de perfil ou muito distantes falham neste critério.

Fórmula:

$IO = ||olho\_direito - olho\_esquerdo||_2$

$IO = \sqrt{((x\_dir - x\_esq)^2 + (y\_dir - y\_esq)^2)}$

Onde:

- olho_direito = coordenadas (x, y) do *landmark* do olho direito
- olho_esquerdo = coordenadas (x, y) do *landmark* do olho esquerdo

Exemplo:

olho_direito = (189, 135)

olho_esquerdo = (231, 138)

$IO = \sqrt{((189 - 231)^2 + (135 - 138)^2)}$

$= \sqrt{((-42)^2 + (-3)^2)}$

$= \sqrt{(1764 + 9)}$

$= \sqrt{1773}$

= 42.1 pixels (≥ 28)

4.3 Nitidez da Imagem (*Blur Detection*)

Critério: *blur_score* ≥ 40.0

Justificativa: Imagens desfocadas (causadas por movimento ou problemas de foco) geram *embeddings* inconsistentes. O operador de *Laplace* detecta a variância de bordas (imagens nítidas têm alta variância).

Fórmula:

$blur\_score = \text{var}(\nabla^2 I)$

Onde:

- I = imagem em escala de cinza da região facial;
- ∇^2 = operador Laplaciano (detecta bordas);
- var() = variância estatística;

Implementação:

```
def variance_of_laplacian(gray_image):
    laplacian = cv2.Laplacian(gray_image, cv2.CV_64F)
    return laplacian.var()
```

Interpretação:

- blur_score < 40: Imagem muito desfocada → **REJEITAR**
- blur_score ≥ 40: Imagem nítida → **ACEITAR**

4.4 Razão

Critério: Detecta expressões faciais extremas;

Fórmula:

$MW = ||canto\_direito\_boca - canto\_esquerdo\_boca||_2$

$canto\_esquerdo = (x_1, y_1)$ $canto\_direito = (x_2, y_2)$

$MW = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$

$razão = MW / IO$

Onde:

- MW = *Mouth Width* é a distância euclidiana entre os dois cantos da boca;
- IO = distância interocular;

Interpretação:

- Razão muito alta (>1.0): boca muito aberta (gritando, bocejando)
- Razão muito baixa (<0.6): boca comprimida ou face lateral
- Razão normal: ≈ 0.85 a 0.95

4.5 Processo de Cadastro

Para aumentar a robustez, o sistema não cadastra com uma única foto. Em vez disso, coleta 12 amostras que passam por todos os critérios de qualidade.

Algoritmo de cadastro:

1. Iniciar coleta (usuário pressiona tecla 0-9)
2. Para cada frame:
   a. Detectar face alvo
   b. Verificar *tamanho_ok AND io_ok AND blur_ok*
   c. Se aprovado: adicionar *embedding* à lista
   d. Atualizar contador (X/12 amostras)
3. Quando 12 amostras coletadas:
   a. Calcular mediana dos 12 *embeddings*
   b. Normalizar *embedding* final
   c. Salvar no banco de dados

Por que mediana?

A mediana é mais robusta que a média, eliminando outliers (amostras atípicas) e capturando a representação mais "central" da face.

5. BIBLIOTECAS E TECNOLOGIAS UTILIZADAS

5.1 *OpenCV (cv2)* - Versão 4.8.0

O que faz: *OpenCV* (*Open Source Computer Vision Library*) é a biblioteca principal do sistema, fornecendo:

- **Captura de vídeo**: Acesso à webcam e controle de resolução/FPS;
- **Carregamento de modelos *ONNX***: Suporte nativo para redes neurais em formato *ONNX;*
- ***Face Detection*** (*FaceDetectorYN*): *Wrapper* para o modelo *YuNet*;
- ***Face Recognition*** (*FaceRecognizerSF*): *Wrapper* para o modelo *SFace*, incluindo alinhamento e extração de *features;*
- ***Object Tracking***: Implementações de *trackers* (*KCF, MOSSE, CSRT*) para seguir faces entre frames;
- **Processamento de imagem**: Conversão de cores, redimensionamento, *blur detection (Laplaciano);*

- **Desenho de *overlays***: Retângulos, linhas, textos para visualização;

Por que escolhemos *OpenCV*?

Atualmente é a biblioteca mais madura e otimizada para visão computacional em tempo real, com ampla documentação e comunidade ativa.

5.2 NumPy - Versão 1.24.3

O que faz: *NumPy* (*Numerical Python*) é a biblioteca de computação numérica do sistema:

- ***Arrays* multidimensionais**: Armazena *embeddings* e imagens de forma eficiente;
- **Operações vetoriais**: Normalização L2, produto escalar, cálculos estatísticos
- **Álgebra linear**: *np.linalg.norm()* para cálculo de normas, *np.dot()* para produto escalar;
- **Operações estatísticas**: Mediana *(np.median())* para agregação de *embeddings;*
- **Performance**: Implementações otimizadas em C, muito mais rápidas que Python puro;

5.3 MediaPipe – Versão 0.10.3

O que faz: *MediaPipe* é um framework do *Google* usado para detecção e rastreamento preciso de *landmarks* faciais 3D, indo muito além dos 5 pontos básicos retornados pelo *YuNet*.

- 468 landmarks 3D — representam com precisão os contornos da face, olhos, sobrancelhas, boca e mandíbula.
- Face Mesh — cria uma malha densa e contínua do rosto, possibilitando medições de proporções faciais (IO, MW, EAR) e rastreamento fino de movimento.
- Tempo real — projetado para rodar eficientemente em CPU e GPU, tanto em desktop quanto em dispositivos móveis.

Uso no projeto: O MediaPipe é utilizado como complemento ao YuNet, adicionando precisão geométrica à análise facial.

Durante a captura, o sistema usa os landmarks do Face Mesh para:

- Calcular IO (*Inner Opening*) e MW (*Mouth Width*) — usados nas métricas de qualidade;
- Extrair posições detalhadas de olhos, boca e contorno facial;
- Gerar polígonos de visualização.

Por que ainda é considerado "opcional":

O pipeline principal (detecção, extração e comparação de *embeddings*) funciona apenas com *YuNet + SFace.*

O *MediaPipe* é um módulo auxiliar, pode ser desativado sem comprometer o reconhecimento facial mas é necessário para funções avançadas como checagem de foco (*blur_ok*), abertura ocular/bucal (*io_ok*) e visualização educativa de *landmarks*.

5.4 Modelos ONNX (YuNet + SFace)

O que é ONNX?

O *ONNX* (*Open Neural Network Exchange*) é um formato aberto e padronizado para representação de modelos de aprendizado profundo. Ele permite que redes neurais treinadas em frameworks diferentes, como *PyTorch*, *TensorFlow* ou *Keras* sejam exportadas e executadas de forma interoperável em qualquer *runtime* compatível.

Vantagem principal: o ONNX elimina a dependência do framework original, permitindo inferência otimizada e multiplataforma.

YuNet

Atua como o primeiro estágio do pipeline — detecta rapidamente a face no frame, identifica seus landmarks principais e fornece as coordenadas exatas para o recorte e alinhamento da região facial.

Propriedade	Descrição
Arquivo	face_detection_yunet_2023mar.onnx *(~8.1 MB)*
Arquitetura base	SSD MobileNet V1
Função principal	Detecção de faces e localização de 5 landmarks (olhos, nariz e boca)
Desempenho	≈ 15–20 ms por frame (CPU)
Saída do modelo	Bounding Box + Coordenadas dos 5 landmarks
Uso no sistema	Define a região de interesse (ROI) facial para alinhamento e extração posterior pelo SFace

SFace

Gera uma assinatura vetorial única da face detectada, que é posteriormente normalizada com L2 e comparada ao banco (db_faces.json) para determinar correspondência de identidade.

Propriedade	Descrição
Arquivo	face_recognition_sface_2021dec.onnx *(~41.6 MB)*
Arquitetura base	ResNet modificada com Sigmoid-Constrained Hypersphere Loss
Função principal	Extração de embeddings faciais de 128 dimensões
Desempenho	≈ 35–45 ms por face (CPU)
Treinamento	Pré-treinado em milhões de identidades para maximizar separabilidade entre classes
Saída do modelo	Vetor de 128 floats representando a identidade facial (embedding)
Uso no sistema	Alimenta o estágio de comparação e decisão via similaridade do cosseno

Integração no Pipeline

Ordem	**Módulo**	**Função**	**Saída**
1	YuNet	Detecta e isola o rosto	Bounding Box + Landmarks (x, y)
2	Pré-processamento	Alinha e recorta a região facial	Face alinhada 128×128
3	SFace	Extrai o embedding da face	Vetor de 128 dimensões
4	Normalização L2 + Cosine	Compara com base de dados	Similaridade e decisão final

5.5 *Requests* - Versão 2.31.0

O que faz: Biblioteca Python para fazer requisições HTTP, usada para integração com a API Flask:

- **POST /api/face-id**: Envia FaceID gerado para o app web;
- **GET /api/face-id/status/{id}**: Verifica se o cadastro foi confirmado no app;
- **GET /api/student/{id}**: Busca informações e pendências do aluno.

Por que é necessário: Permite que o sistema desktop e o aplicativo web comuniquem-se, sincronizando cadastros e exibindo informações contextuais (nome completo, pendências, etc.).

5.6 *Python Collections* (deque, Counter)

O que faz: Estruturas de dados nativas do Python usadas para votação temporal:

- **deque** (double-ended queue): Fila circular com tamanho máximo fixo, ideal para buffer deslizante de 25 frames
- **Counter**: Conta ocorrências de elementos, usado para decidir voto majoritário

5.7 JSON (Python Standard Library)

O que faz: Módulo nativo do Python para serialização/deserialização de dados em formato JSON:

- **Salvar banco de dados**: *Embeddings* e *IDs* são salvos em *db_faces.json;*
- **Formato legível**: *JSON* permite inspeção manual dos vetores para fins educacionais.

Estrutura do banco:

```
{
  "A3F2": [-0.1623, 0.0234, 0.1689, ..., -0.0812],
  "B5C1": [0.2341, -0.4123, 0.0891, ..., 0.5623],
  "DC40": [-0.0812, 0.5623, -0.3241, ..., 0.7123]
}
```

Limitação: Busca linear O(n) - adequado para centenas de faces, não para milhares. Para escala maior, usar banco vetorial (*FAISS*, *Milvus*).

6. CONCLUSÃO

O FaceMap PRO alcançou seu objetivo de tornar visível e compreensível o funcionamento matemático de sistemas de reconhecimento facial baseados em *deep learning*. Através da visualização em tempo real dos *embeddings* de 128 dimensões, cálculos de similaridade de cosseno e métricas de qualidade, o sistema desmistifica conceitos abstratos de inteligência artificial.

Nota: *Este trabalho tem caráter exclusivamente didático, demonstrando conceitos de visão computacional e aprendizado de máquina. O sistema não é adequado para aplicações de segurança crítica sem validações rigorosas.*

REFERÊNCIAS

OPENCV FOUNDATION. *OpenCV – Open Source Computer Vision Library.* Disponível em: https://docs.opencv.org/4.x/. Acesso em: 09 out. 2025.

OPENCV FOUNDATION. *YuNet – Face Detection Model.* Disponível em: https://github.com/opencv/opencv_zoo/tree/main/models/face_detection_yunet. Acesso em: 09 out. 2025.

OPENCV FOUNDATION. *SFace – Face Recognition Model.* Disponível em: https://github.com/opencv/opencv_zoo/tree/main/models/face_recognition_sface. Acesso em: 09 out. 2025.

PYTHON SOFTWARE FOUNDATION. *Python 3 Standard Library Documentation.* Disponível em: https://docs.python.org/3/. Acesso em: 09 out. 2025.

NUMPY DEVELOPERS. *NumPy: Fundamental Package for Scientific Computing with Python.* Disponível em: https://numpy.org/doc/stable/. Acesso em: 09 out. 2025.

ONNX COMMUNITY. *Open Neural Network Exchange (ONNX) – Model Interoperability Standard.* Disponível em: https://onnx.ai/. Acesso em: 09 out. 2025.